Parte 3

I package standard

IN QUESTA PARTE

✔ Capitolo 8: Panoramica sui package standard 187

✔ Capitolo 9: Il package del linguaggio 203

✔ Capitolo 10: Il package delle utilità 235

✔ Capitolo 11: Il package di input e output 271

✔ Capitolo 12: Il package di rete 295

✔ Capitolo 13: Il package dell'AWT 311

✔ Capitolo 14: Il package del testo 343

✔ Capitolo 15: Il package della sicurezza 367

✔ Capitolo 16: Il package RMI 389

✔ Capitolo 17: Il package della riflessione 417

✔ Capitolo 18: Il package di SQL 441

CAPITOLO 8

Panoramica sui package standard

di Michael Morrison

IN QUESTO CAPITOLO

- ✔ Il package del linguaggio 188
- ✔ Il package delle utilità 191
- ✔ Il package di input e output 192
- ✔ Il package di rete 194
- ✔ Il package delle finestre (AWT) 196
- ✔ Il package del testo 198
- ✔ Il package della sicurezza 199
- ✔ Il package dell'RMI 200
- ✔ Il package della riflessione 200
- ✔ Il package di SQL 200
- ✔ Riepilogo 201

Il riutilizzo del codice è uno dei vantaggi principali della progettazione orientata agli oggetti. La creazione di classi riutilizzabili e da cui è possibile derivare altre classi permette di risparmiare tempo ed energie, migliorando notevolmente la produttività. Java stesso sfrutta questa possibilità di riutilizzo del codice per l'implementazione di una grande varietà di oggetti standard disponibili per i programmatori. Gli oggetti standard di Java sono conosciuti collettivamente con il nome di *package standard.*

I package standard di Java contengono gruppi di classi correlate, le interfacce e le definizioni delle eccezioni e degli errori. In Java sono inclusi dieci package standard: il package del linguaggio, delle utilità, delle operazioni di I/O, di rete, delle finestre, del testo, della sicurezza, dell'RMI, della riflessione e di SQL. In questo capitolo vengono descritti tutti i package, le classi e le interfacce incluse in ognuno di essi.

Il package del linguaggio

Questo package, `java.lang`, contiene classi che costituiscono il cuore del linguaggio Java e che si trovano al livello più basso dei package standard di Java. Ad esempio, la classe `Object`, da cui derivano tutte le classi, si trova in questo package.

È impossibile scrivere programmi Java senza avere a che fare con almeno alcuni elementi del package del linguaggio; nel prossimo capitolo vengono fornite ulteriori informazioni sul funzionamento interno di questo package. Le classi più importanti contenute nel package sono le seguenti.

- ✔ La classe `Object`.
- ✔ Le classi involucro dei tipi di dati.
- ✔ La classe `Math`.
- ✔ Le classi `String`.
- ✔ Le classi `System` e `Runtime`.
- ✔ Le classi dei thread.
- ✔ Le classi di gestione delle classi.
- ✔ Le classi di gestione delle eccezioni.
- ✔ La classe `Process`.

Il package della riflessione di Java, `java.lang.reflect`, descritto in questo capitolo e nel Capitolo 17, dal punto di vista tecnico è un sottopackage di `java.lang`. Tuttavia, a causa della sua importanza per l'architettura generale di Java, in questo libro viene trattato come un package autonomo.

La classe Object

La classe `Object` è la superclasse di tutte le classi di Java. Poiché tutte le classi derivano da `Object`, i metodi definiti in `Object` sono condivisi da tutte le classi; ciò significa che vi è una serie di metodi che tutte le classi di Java supportano. `Object` include metodi per creare copie degli oggetti, per verificarne l'uguaglianza e per convertire il valore di un oggetto in stringa.

Le classi involucro dei tipi di dati

I tipi di dati primitivi di Java (`int`, `char`, `float` e così via) non sono implementati come classi. Tuttavia, spesso è utile conoscere più informazioni su un tipo primitivo, e non solo il suo valore. Implementando *classi involucro* per i tipi fondamentali, è possibile mantenere ulteriori informazioni e definire metodi che operano sui tipi. Le classi involucro dei tipi di dati fungono da versioni di classe dei tipi di dati primitivi e hanno nomi simili a quelli dei tipi per i quali agiscono da involucro; ad esempio, l'involucro per `int` è la classe `Integer`. Di seguito sono elencate le classi involucro dei tipi di dati di Java.

- `Boolean`
- `Character`
- `Double`
- `Float`
- `Integer`
- `Long`

Gli involucri dei tipi sono utili anche perché per i parametri di molte classi di utilità di Java sono necessarie classi vere e proprie, anziché tipi semplici. Vale la pena notare che gli involucri dei tipi di dati e i tipi semplici non sono intercambiabili; tuttavia, è possibile ottenere un tipo semplice da un involucro tramite una chiamata di metodo, come descritto nel prossimo capitolo.

La classe Math

La classe `Math` raggruppa costanti e funzioni matematiche. È interessante notare che tutte le variabili e tutti i metodi in `Math` sono statici e che la classe `Math` stessa è finale. Ciò significa che non è possibile derivare nuove classi da `Math` e che non è possibile creare istanze da questa classe. È consigliabile pensare alla classe `Math` come a una conglomerazione di metodi e di costanti che permettono di eseguire calcoli matematici.

La classe `Math` tra l'altro include le costanti `E` e `PI` e metodi per determinare il valore assoluto di un numero, per calcolare funzioni trigonometriche e per determinare il valore minimo e il massimo.

Le classi String

Per diversi motivi, principalmente correlati alla sicurezza, Java implementa le stringhe di testo come classi, anziché obbligare il programmatore a utilizzare array di caratteri. Le due classi di Java che rappresentano le stringhe sono `String` e `StringBuffer`; la prima è utile per operare con stringhe costanti di cui non può essere modificato il valore o la lunghezza, mentre la seconda viene utilizzata con stringhe di valore e lunghezza variabili.

Le classi System e Runtime

Le classi `System` e `Runtime` costituiscono un mezzo per accedere alle risorse del sistema e dell'ambiente di esecuzione. Come la classe `Math`, la classe `System` è finale ed è composta solamente da variabili e da metodi statici. La classe `System` in sostanza è un'interfaccia di programmazione indipendente dal sistema per le risorse del sistema stesso, ad esempio per i dispositivi di input e di output standard, `System.in` e `System.out`, che di norma creano un modello per la tastiera e per il monitor.

La classe `Runtime` fornisce l'accesso diretto all'ambiente di esecuzione. Un esempio di routine di esecuzione è il metodo `freeMemory()`, che restituisce la quantità di memoria libera.

Le classi dei thread

Java è un ambiente multithreading e include diverse classi per la gestione dei thread e per operare con essi. Di seguito sono presentate le classi e le interfacce utilizzate con i programmi multithreading.

- ✔ `Thread`: utilizzata per creare un thread di esecuzione in un programma.
- ✔ `ThreadDeath`: utilizzata per eliminare un thread dopo che ha terminato l'esecuzione.
- ✔ `ThreadGroup`: utilizzata per organizzare un gruppo di thread.
- ✔ `Runnable`: costituisce uno strumento alternativo per creare un thread senza creare sottoclassi della classe `Thread`.

I thread e il multithreading sono discussi in dettaglio nel Capitolo 5.

Le classi di gestione delle classi

Java include due classi per gestire le classi: `Class` e `ClassLoader`; la prima fornisce informazioni di esecuzione su una classe, ad esempio il nome, il tipo e la superclasse, ed è utile per richiedere a una classe informazioni di esecuzione, ad esempio il nome. La classe `ClassLoader` viene utilizzata per caricare classi nell'ambiente di esecuzione da un file o per caricare classi distribuite in una connessione di rete.

Le classi di gestione delle eccezioni

La gestione degli errori di esecuzione è uno strumento molto importante in qualsiasi ambiente di programmazione. Per gestire gli errori di esecuzione, Java include le seguenti classi.

- ✔ `Throwable`: fornisce le funzionalità per la gestione degli errori a basso livello, ad esempio un elenco dello stack di esecuzione.
- ✔ `Exception`: derivata da `Throwable`, fornisce il livello base di funzionalità di tutte le classi di eccezioni definite nel sistema di Java. Utilizzata per gestire errori normali.
- ✔ `Error`: derivata da `Throwable` (come la classe `Exception`), viene però utilizzata per gestire errori anormali e imprevisti. Pochissimi programmi di Java utilizzano la classe `Error`; per gestire gli errori di esecuzione, la maggior parte dei programmi utilizza invece la classe `Exception`.

La gestione degli errori con le eccezioni è discussa in dettaglio nel Capitolo 6.

La classe Process

Java supporta processi di sistema con un'unica classe, `Process`, che rappresenta i processi generici del sistema che vengono creati quando si utilizza la classe `Runtime` per eseguire i comandi di sistema.

Il package delle utilità

Questo package, `java.util`, contiene varie classi che eseguono diverse funzioni di utilità; tra le altre vi sono una classe per lavorare con le date, una serie di classi di struttura dei dati, una classe per generare numeri casuali e una classe per la suddivisione in token delle stringhe. Nel Capitolo 10 vengono discusse in modo più approfondito le classi che compongono il package delle utilità; le più importanti sono le seguenti.

- ✔ La classe `Date`.
- ✔ Le classi di struttura dei dati.
- ✔ La classe `Random`.
- ✔ La classe `StringTokenizer`.
- ✔ La classe `Properties`.
- ✔ Le classi di osservatori.

La classe Date

La classe `Date` rappresenta data e ora in modo indipendente dal sistema. Questa classe contiene i metodi per richiamare la data e l'ora corrente e per calcolare i giorni della settimana e del mese.

Le classi di struttura dei dati

Le classi e le interfacce di struttura dei dati di Java implementano le comuni strutture per memorizzare i dati.

- ✔ `BitSet`: rappresenta una serie di bit, o *campo di bit*.
- ✔ `Dictionary`: classe astratta che costituisce un meccanismo di ricerca per la mappatura delle chiavi in valori.
- ✔ `Hashtable`: derivata da `Dictionary`, fornisce ulteriore supporto per le operazioni con le chiavi e con i valori.
- ✔ `Properties`: derivata da `Hashtable`, fornisce ulteriori funzionalità per la lettura e la scrittura su dispositivi di input e output.
- ✔ `Vector`: implementa un array in grado di crescere dinamicamente.
- ✔ `Stack`: derivata da `Vector`, implementa uno stack classico di oggetti LIFO (Last-In-First-Out, l'ultimo che entra è il primo a uscire).
- ✔ `Enumeration`: questa interfaccia specifica una serie di metodi per contare (iterare) una serie di valori.

La classe Random

Molti programmi, in particolare quelli che hanno come modello il mondo reale, necessitano di un certo livello di casualità. In Java la causalità si ottiene per mezzo della classe `Random`, che implementa un generatore di numeri casuali che fornisce un flusso di numeri pseudo-casuali. Un programma per una slot machine è un buon esempio di programma che può utilizzare la classe `Random`.

La classe StringTokenizer

La classe `StringTokenizer` costituisce uno strumento per convertire stringhe di testo in singoli token. Specificando una serie di delimitatori, è possibile suddividere le stringhe di testo in token utilizzando la classe `StringTokenizer`. La suddivisione in token delle stringhe è utile in diversi programmi, dai compilatori ai giochi di avventura che si basano sul testo.

Le classi di osservatori

Il paradigma di visualizzazione di modelli sta diventando sempre più popolare nella programmazione orientata agli oggetti. Questo modello divide un programma in dati e visualizzazioni dei dati, e in Java è supportato tramite la classe `Observable` e l'interfaccia `Observer`. La prima viene derivata per definire i dati osservabili in un programma, che vengono quindi connessi a una o più classi di osservatori. Queste sono implementazioni dell'interfaccia `Observer`. Quando un oggetto `Observable` modifica il suo stato, informa tutti i suoi osservatori del cambiamento.

Il package di input e output

Questo package, `java.io`, contiene classi che supportano la lettura e la scrittura di dati in e da diversi dispositivi di input e di output, inclusi i file. Il package di I/O include le classi per l'input e l'output di flussi di dati, per le operazioni con i file e per la suddivisione in token di flussi di dati. Nel Capitolo 11 sono fornite ulteriori informazioni sulle classi che compongono il package di I/O; le più importanti sono le seguenti.

- ✔ Le classi del flusso di input.
- ✔ Le classi del flusso di output.
- ✔ Le classi dei file.
- ✔ La classe `StreamTokenizer`.

Le classi del flusso di input

Java utilizza i flussi di input per gestire la lettura di dati da una *sorgente di input*, che può essere un file, una stringa, la memoria o qualsiasi altra cosa che contenga dati. Le classi del flusso di input sono le seguenti.

- ✔ `InputStream`
- ✔ `BufferedInputStream`
- ✔ `ByteArrayInputStream`
- ✔ `DataInputStream`
- ✔ `FileInputStream`
- ✔ `FilterInputStream`
- ✔ `LineNumberInputStream`
- ✔ `PipedInputStream`
- ✔ `PushbackInputStream`
- ✔ `SequenceInputStream`
- ✔ `StringBufferInputStream`

`InputStream` è una classe astratta che serve come classe di base per tutti i flussi di input; tra l'altro definisce un'interfaccia per la lettura dei flussi di byte di dati, trovando il numero di byte disponibili per la lettura e muovendo il puntatore alla posizione del flusso. Tutti gli altri flussi di input forniscono il supporto per la lettura dei dati da diversi tipi di dispositivi di input.

Le classi del flusso di output

I flussi di output sono la controparte dei flussi di input e gestiscono la scrittura di dati in una *sorgente di output*. Come le sorgenti di input, le sorgenti di output includono file, stringhe, la memoria e qualsiasi cosa che possa contenere dati. Le classi del flusso di output definite in `java.io` sono le seguenti.

- ✔ `OutputStream`
- ✔ `BufferedOutputStream`
- ✔ `ByteArrayOutputStream`
- ✔ `DataOutputStream`
- ✔ `FileOutputStream`
- ✔ `FilterOutputStream`
- ✔ `PipedOutputStream`
- ✔ `PrintStream`

La classe `OutputStream` è una classe astratta che serve come classe di base per tutti i flussi di output. `OutputStream` definisce un'interfaccia per la scrittura di flussi di byte di dati in una sorgente di output. Tutti gli altri flussi di output forniscono il supporto per la scrittura di

dati in dispositivi di output diversi. I dati scritti da un flusso di output sono formattati in modo da poter essere letti da un flusso di input.

Le classi dei file

I file sono il metodo maggiormente utilizzato per memorizzare i dati nei sistemi di computer. Java supportata file con due diverse classi: `File` e `RandomAccessFile`. La classe `File` fornisce un'astrazione per i file, che tiene in considerazione le funzionalità dipendenti dal sistema dell'account; mantiene le informazioni su un file, inclusa la locazione in cui è memorizzato e come è possibile accedervi. La classe `File` non ha metodi per la lettura e la scrittura di dati da e in un file, è utile solo per modificare gli attributi dei file e per richiedere informazioni su di essi. In realtà, è possibile pensare ai dati della classe `File` come a rappresentazioni di un nome di file e ai metodi della classe come a rappresentazioni di comandi del sistema operativo che agiscono sui nomi di file.

La classe `RandomAccessFile` include numerosi metodi per scrivere e leggere dati da e in un file. `RandomAccessFile` contiene molti metodi per leggere e scrivere diversi tipi di informazioni, vale a dire gli involucri dei tipi di dati.

La classe StreamTokenizer

La classe `StreamTokenizer` permette di convertire un flusso di input di dati in un flusso di token e contiene una serie di metodi per definire la sintassi lessicale dei token. La suddivisione dei flussi in token è utile per l'analisi dei flussi di dati testuali.

Il package di rete

Questo package, `java.net`, contiene classi che permettono di eseguire una vasta gamma di comunicazioni di rete. Esso include il supporto specifico per gli URL, per i socket TCP, per gli indirizzi IP e per i socket UDP. Le classi di rete di Java rendono semplice e diretta l'implementazione di soluzioni Internet client/server in Java. Nel Capitolo 12 sono fornite ulteriori informazioni sulle classi del package di rete. Le classi incluse in questo package sono le seguenti.

- ✔ La classe `InetAddress`.
- ✔ Le classi degli URL.
- ✔ Le classi dei socket.
- ✔ La classe `ContentHandler`.

La classe InetAddress

La classe `InetAddress` *crea un modello di indirizzo IP e contiene metodi per ottenere* informazioni sull'indirizzo, ad esempio per richiamare il testo del nome o la pura rappresentazio-

ne IP dell'host a cui corrisponde l'indirizzo. InetAddress contiene inoltre metodi statici che permettono di scoprire informazioni sugli host senza dover effettivamente creare un oggetto InetAddress.

Le classi degli URL

Le classi degli URL vengono utilizzate per rappresentare e interagire con gli URL (Uniform Resource Locator),che sono riferimenti a informazioni nel Web. Le classi degli URL incluse nel package di rete sono le seguenti.

- ✔ URL: rappresenta un URL. Gli oggetti URL sono costanti, vale a dire che il loro valore non può essere modificato dopo che sono stati creati. In questo modo, gli oggetti URL rappresentano più da vicino gli URL fisici, che sono anch'essi costanti.
- ✔ URLConnection: classe astratta che definisce la struttura necessaria per effettuare una connessione per mezzo di un URL. Questa classe deve essere derivata per fornire le funzionalità per un tipo specifico di connessione URL.
- ✔ URLStreamHandler: classe astratta che definisce il meccanismo necessario per aprire flussi che si basano su URL.
- ✔ URLEncoder: permette di convertire una stringa di informazioni testuali in un formato adatto per la comunicazione per mezzo di un URL.

Le classi dei socket

Le classi dei socket sono forse le più importanti del package di rete. Forniscono l'intera struttura per eseguire le comunicazioni di rete per mezzo di approcci diversi. Le classi che forniscono il supporto per i socket in Java sono le seguenti.

- ✔ SocketImpl: classe astratta che definisce un livello base di funzionalità necessario per tutti i socket. Queste funzionalità includono sia variabili membro sia una raccolta essenziale di metodi. *Le implementazioni specifiche dei socket derivano da* SocketImpl.
- ✔ Socket: fornisce il supporto per i socket di flussi dal lato client, che sono socket che comunicano in tempo reale tramite una connessione con un alto grado di affidabilità.
- ✔ ServerSocket: utilizzata per implementare il lato server del supporto per i socket di flussi.
- ✔ DatagramSocket: contiene tutto quanto è necessario per eseguire le comunicazioni di *socket di datagrammi*, che sono socket che inviano pacchetti di informazioni con un basso livello di affidabilità e di precisione temporale. A differenza dei socket di flussi, i socket di datagrammi non si basano su una connessione dal vivo, vale a dire che inviano e ricevono i dati quando risulta più conveniente. I dati che vengono trasferiti con un socket di datagrammi devono essere incapsulati da un oggetto DatagramPacket.
- ✔ DatagramPacket: contiene informazioni critiche per un pacchetto di informazioni che viene trasferito con un socket di datagrammi.

La classe ContentHandler

La classe `ContentHandler` serve da struttura per la gestione di diversi tipi di dati Internet; ad esempio, è possibile scrivere un gestore di contenuto per elaborare e visualizzare il formato di un file proprietario. Per farlo, si deriva una classe da `ContentHandler` e si scrive il codice per creare un oggetto da un flusso di dati che rappresenta il tipo dell'oggetto.

Il package delle finestre (AWT)

Questo package, `java.awt`, è composto da una serie di classi che forniscono un'ampia gamma di funzionalità per la grafica e per l'interfaccia utente. Sono incluse classi che rappresentano gli elementi dell'interfaccia grafica, quali le finestre, le finestre di dialogo, i menu, i pulsanti, le caselle di controllo e i campi di testo e anche elementi grafici generali, quali i caratteri. Nel Capitolo 13 sono fornite ulteriori informazioni sul package delle finestre. Le classi più importanti incluse in questo package sono le seguenti.

- ✔ Le classi grafiche.
- ✔ Le classi dei gestori di layout.
- ✔ Le classi `Font`.
- ✔ Le classi delle dimensioni.
- ✔ La classe `MediaTracker`.

Il package delle finestre viene spesso detto Abstract Windowing Toolkit (AWT), da cui deriva il nome `java.awt`.

Le classi grafiche

Le classi grafiche svolgono tutte una particolare funzione relativa all'input grafico dell'utente o alla visualizzazione; per questo motivo sono spesso indispensabili nella programmazione degli applet. Queste classi includono il supporto per qualsiasi componente grafico, dalle caselle di controllo ai menu, ai canvas, alla rappresentazione dei colori.

Nonostante siano state menzionate solo in relazione agli applet di Java, le classi grafiche sono ugualmente utili per le applicazioni autonome di Java. Basta considerare che per le applicazioni autonome è necessario creare una cornice in cui inserire tutti gli elementi grafici, mentre per gli applet è sufficiente utilizzare la finestra assegnata all'applet nella pagina Web che lo contiene.

Una delle classi più importanti è `Graphics`, una classe generica per l'output grafico in grado di eseguire qualsiasi tipo di funzione di disegno. La classe `Graphics` è responsabile di tutto l'output grafico generato dagli applet di Java.

Un'altra classe grafica fondamentale è `Component`, utilizzata come genitore per molte altre classi grafiche, principalmente perché fornisce la struttura necessaria per gli elementi grafici di base.

Le classi dei gestori di layout

Le classi dei gestori di layout forniscono una struttura per il controllo della disposizione fisica degli elementi della GUI. Ad esempio, se si desidera disporre una riga di pulsanti in un certo modo lungo il bordo superiore della finestra di un applet, si utilizza un gestore di layout. Le classi dei gestori di layout implementate nel package delle finestre sono elencate di seguito.

- ✔ `BorderLayout`: dispone gli elementi grafici, chiamati anche *componenti*, lungo un bordo della finestra, con un elemento in ogni posizione (nord, sud, est, ovest e centro).
- ✔ `CardLayout`: dispone i componenti uno sopra l'altro, come in un mazzo di carte; è possibile modificare l'ordine per visualizzare componenti diversi.
- ✔ `FlowLayout`: dispone i componenti da sinistra a destra nella finestra; quando non è più possibile aggiungere componenti in una riga, ne viene iniziata una nuova.
- ✔ `GridLayout`: dispone componenti di dimensioni uguali in una griglia con un numero specifico di righe e di colonne.
- ✔ `GridBagLayout`: simile alla classe `GridLayout`, a eccezione del fatto che i componenti in `GridBagLayout` non devono avere uguali dimensioni; questo layout fornisce al programmatore una grande flessibilità.

Tutte le classi dei layout derivano dall'interfaccia `LayoutManager`, che definisce le funzionalità di base necessarie per i gestori di layout grafici.

Le classi dei caratteri

Le classi dei caratteri sono composte dalla classe `Font` e dalla classe `FontMetrics`. La prima rappresenta oggetti carattere grafico con attributi quali il nome, le dimensioni e lo stile; gli oggetti `Font` possono inoltre essere in *corsivo* o in **grassetto**. La seconda classe permette di trovare le informazioni sulle dimensioni di un carattere; ad esempio, è possibile utilizzare un oggetto `FontMetrics` per trovare l'altezza di un carattere o caratteristiche più specifiche, come la spaziatura delle righe.

Le classi delle dimensioni

Le classi delle dimensioni rappresentano un modo utile per specificare diverse dimensioni grafiche in Java. Nel package delle finestre sono definite le seguenti classi.

- ✔ `Dimension`: rappresenta le dimensioni di base rettangolari di un elemento grafico. La classe include due variabili membro pubbliche per memorizzare la larghezza e l'altezza di un oggetto rettangolare.
- ✔ `Rectangle`: simile a `Dimension`, con l'eccezione del fatto che `Rectangle` include anche le coordinate x e y dell'angolo superiore sinistro di un elemento grafico. In altre parole, questa classe mantiene le dimensioni di un elemento grafico e anche la sua posizione.

- Point: simile alla classe Rectangle, a eccezione del fatto che mantiene solo una posizione x,y.
- Polygon: rappresenta un poligono, che in sostanza è una serie di punti connessi.

La classe MediaTracker

La classe MediaTracker permette di rilevare quando è terminata la trasmissione di risorse multimediali in una connessione di rete. Al momento la classe MediaTracker supporta solo il rilevamento di immagini, ma senza dubbio in una futura versione di Java verrà aggiunto il supporto per i suoni e per altri elementi multimediali, a mano a mano che diventeranno più popolari. La classe MediaTracker è molto utile per gli applet, in quanto permette loro di sapere quando una particolare immagine è pronta per essere visualizzata, cosa critica in alcuni casi.

Il package del testo

Questo package, java.text, è una parte essenziale del supporto di Java per l'internazionalizzazione. Il package del testo contiene classi e interfacce che permettono di gestire il testo specifico per una località particolare. In altre parole, il package fornisce il meccanismo di base che permette la mappatura del testo in base a un linguaggio o a una regione specifica. Le classi e le interfacce definite in questo package si basano sulla codifica dei caratteri Unicode 2.0 e possono essere utilizzate per adattare testo, numeri, date, valuta e oggetti definiti dall'utente alle convenzioni di qualsiasi paese. Nel Capitolo 14 sono fornite ulteriori informazioni sulle classi e sulle interfacce nel package del testo. Alcune delle classi più importanti incluse in questo package sono le seguenti.

- Le classi di formattazione.
- La classe Collator.
- La classe TextBoundary.

Le classi di formattazione

Il package del testo contiene diverse classi per la formattazione dei dati in base a diverse convenzioni culturali. Queste classi possono inoltre riportare le stringhe formattate nella loro forma originale. La classe Format è una classe di base astratta che incapsula la formattazione e l'analisi delle stringhe in base alla località. Da Format derivano tre sottoclassi principali: NumberFormat, DateFormat e MessageFormat, utilizzate per formattare rispettivamente numeri, date e messaggi. Da queste classi derivano ulteriori classi di formattazione più specifiche.

La classe Collator

La classe Collator fornisce il supporto per il confronto di stringhe in base alla località. Utilizzando questa classe, è possibile confrontare due stringhe e valutare le convenzioni

speciali specifiche per una località utilizzate per l'attribuzione di nomi e per la formattazione. Gli strumenti di internazionalizzazione di Java necessitano di questa capacità per ordinare e ricercare il testo in base alle convenzioni specifiche per una località.

La classe TextBoundary

La classe `TextBoundary` determina confini diversi nel testo, ad esempio i confini di parola, di riga e di frase. Questa funzionalità è critica per la gestione di lingue diverse e consente ai programmatori di gestire i ritorni a capo e la selezione del testo automatici.

Il package della sicurezza

Questo package, `java.security`, contiene le funzionalità per incorporare la sicurezza tramite crittografia nelle applicazioni che si basano su Java. La struttura della crittografia nel package della sicurezza include il supporto per la crittografia DSA ed è progettata in modo che successivamente possano essere aggiunti nuovi algoritmi senza difficoltà. Ad esempio, nonostante la crittografia DSA sia l'unico algoritmo di firma digitale integrato nella versione 1.1 di Java, l'API può includere senza problemi altri algoritmi, quali l'RSA, senza che siano necessarie modifiche significative del codice. Nel Capitolo 15 sono fornite ulteriori informazioni sulle classi e sulle interfacce nel package della sicurezza. Alcune delle classi più importanti incluse in questo package sono le seguenti.

- ✔ Le classi delle firme digitali.
- ✔ La classe `MessageDigest`.
- ✔ Le classi di gestione delle chiavi.

Le classi delle firme digitali

Le classi delle firme digitali nel package della sicurezza forniscono il supporto per generare coppie di chiavi pubbliche/private e per firmare e verificare dati digitali arbitrari. Le firme digitali sono utilizzate per autenticare e assicurare l'integrità di dati digitali. La classe `Signature` viene utilizzata per fornire la funzionalità di un algoritmo di firma digitale, quale il DSA: La classe `Identity` rappresenta le *identità*: oggetti del mondo reale come persone, società od organizzazioni le cui identità possono essere autenticate utilizzando le loro chiavi pubbliche. La classe `Signer` rappresenta un'identità in grado anche di firmare digitalmente i dati.

La classe MessageDigest

La classe `MessageDigest` fornisce le funzionalità di un algoritmo per *digest di messaggi*, ad esempio MD5 o SHA. I *digest di messaggi* sono funzioni di hash sicure e unidirezionali che utilizzano dati di dimensioni arbitrarie e producono valori di hash in output di lunghezza fissa. I digest sono utili per produrre "impronte digitali" di dati, che spesso vengono utilizzate nelle firme digitali e in altre applicazioni che necessitano di identificatori unici e non falsificabili dei dati digitali.

Le classi di gestione delle chiavi

Le classi di gestione delle chiavi incluse nel package della sicurezza vengono utilizzate per gestire le chiavi pubbliche e private utilizzate nel processo di firma digitale. `Key` è l'interfaccia di livello superiore per tutte le chiavi e definisce le funzionalità condivise da tutti gli oggetti chiave. La classe `KeyPair` è un semplice contenitore per una coppia di chiavi costituita da una chiave pubblica e da una chiave privata. Infine, la classe `KeyPairGenerator` viene utilizzata per generare coppie di chiavi pubbliche e private.

Il package dell'RMI

Questo package, `java.rmi`, permette agli sviluppatori di creare applicazioni distribuite che si basano sulla chiamata a metodi remoti. Quando si utilizza l'RMI (Remote Method Invocation), si possono richiamare metodi di oggetti di Java remoti da altre macchine virtuali di Java, perfino in host diversi. Il supporto per l'RMI fornito in questo package utilizza la serializzazione degli oggetti per elaborare parametri che vengono passati tra i metodi. Nel Capitolo 16 vengono fornite ulteriori informazioni sul package dell'RMI.

Il package della riflessione

Questo package, `java.lang.reflect`, permette al codice di Java di esaminare e trovare informazioni dettagliate sulla struttura delle classi durante l'esecuzione. Più specificatamente, la riflessione può essere utilizzata per scoprire informazioni sui campi, sui metodi e sui costruttori delle classi. Questo package è utile per le applicazioni che richiedono l'accesso ai membri pubblici di un oggetto di destinazione oppure ai membri dichiarati da una determinata classe. Nel Capitolo 17 sono fornite ulteriori informazioni sul package della riflessione.

JavaBeans si basa in gran parte sull'API della riflessione per consentire ai costruttori di applicazioni di determinare le proprietà esportate dei componenti.

Il package di SQL

Questo package, `java.sql`, fornisce la struttura necessaria agli sviluppatori per scrivere applicazioni di database in grado di eseguire query SQL. Il package di SQL a volte viene detto anche JDBC (Java Database Connectivity). L'SQL è il linguaggio di interrogazione standard per accedere e manipolare i database. Il package di SQL rende possibile l'interazione di un'applicazione Java con qualsiasi database relazionale tramite SQL. L'unico requisito è che il database abbia un driver SQL. La possibilità di accedere a diversi database in modo coerente con Java è un grande passo in avanti per gli sviluppatori di Java. Ad esempio, il package di SQL permette di pubblicare una pagina Web contenente un applet che utilizza informazioni ottenute da un database remoto. Anche le applicazioni di database delle intranet ottengono molti vantaggi dal package di SQL. Nel Capitolo 18 sono fornite ulteriori informazioni sul package di SQL; alcune delle classi e delle interfacce più importanti incluse in questo package sono le seguenti.

- ✔ La classe `DriverManager`.
- ✔ L'interfaccia `Connection`.
- ✔ Le interfacce `Statement` e `ResultSet`.

La classe DriverManager

La classe `DriverManager` fornisce la struttura per gestire una serie di driver JDBC, cosa necessaria per stabilire una connessione con un particolare database. Quando viene inizializzata, la classe `DriverManager` cerca di caricare i driver identificati nelle impostazioni del sistema e di selezionare e utilizzare uno di questi driver quando l'utente cerca di stabilire una connessione a un database.

L'interfaccia Connection

L'interfaccia `Connection` definisce le funzionalità necessarie per una connessione a un database. Quando viene stabilita una connessione a un database per mezzo della classe `DriverManager`, all'utente viene restituito un oggetto derivato da `Connection`, che viene utilizzato come contesto tramite il quale vengono costruite ed eseguite le istruzioni SQL e restituiti i risultati.

Le interfacce Statement e ResultSet

L'interfaccia `Statement` definisce le funzionalità necessarie per un'istruzione SQL da eseguire in un database. Le istruzioni SQL vengono sempre emesse tramite un particolare contesto della connessione. I risultati dell'istruzione SQL eseguita vengono restituiti in un oggetto derivato da `ResultSet`. In base alla gestione dei risultati dei database relazionali standard, l'interfaccia `ResultSet` fornisce l'accesso a una serie di dati dei risultati sotto forma di tabella. L'interfaccia `ResultSet` viene utilizzata per accedere a questi dati riga per riga.

Riepilogo

Questo capitolo ha presentato una panoramica sui dieci package standard di Java: il package del linguaggio, delle utilità, di I/O, di rete, delle finestre, del testo, della sicurezza, dell'RMI, della riflessione e di SQL. Nonostante non siano stati forniti molti dettagli e non sia stato spiegato come utilizzare queste classi in un programma vero, si dovrebbe avere ora un'idea generale dello scopo di questi package. I package standard di Java contengono una ricca serie di classi per superare numerosi ostacoli durante la programmazione.

Ora il lettore dovrebbe avere un'idea di quali classi standard possono essere utilizzate nei programmi di Java e di quali classi devono essere implementate dal programmatore.

Dopo aver visto le funzionalità fornite da questi package, si può passare a vedere come vengono utilizzate le classi incluse in ogni package. I prossimi dieci capitoli si concentrano sui package standard di Java.